LESSONS FROM ALPHAZERO FOR OPTIMAL, MODEL PREDICTIVE, AND ADAPTIVE CONTROL

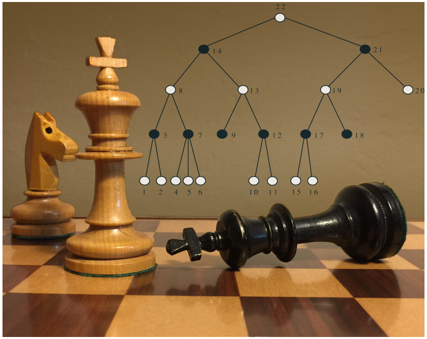
Dimitri P. Bertsekas
Arizona State University

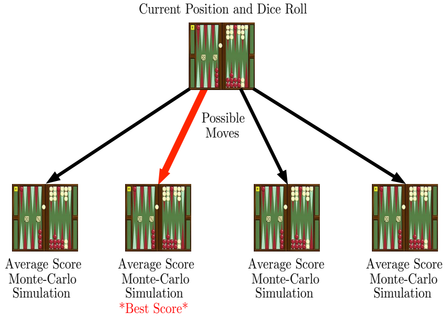
Lecture at MIT on Oct. 18, 2022

Highlights of my recent “Lessons ..." book, Spring 2022

Based on analysis from my books
Rollout, Policy Iteration, and Distributed Reinforcement Learning, 2020
Reinforcement Learning and Optimal Control 2019
Abstract Dynamic Programming, 2013 (3nd Edition, 2022)

Chess and Backgammon - Off-Line Training and On-Line Play





Current Position and Dice Roll
Possible Moves
Average Score Monte-Carlo Simulation
Average Score Monte-Carlo Simulation *Best Score*
Average Score Monte-Carlo Simulation
Average Score Monte-Carlo Simulation

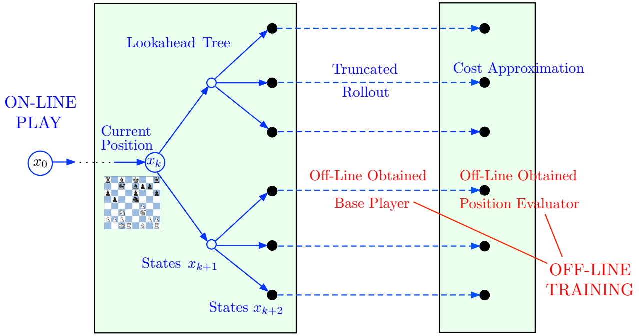
Both AlphaZero (2017) and TD-Gammon (1996) involve two algorithms:

- Off-line training of value and/or policy neural network approximations
- On-line play by multistep lookahead, rollout, and cost function approximation

Strong connections to DP, policy iteration, and RL-type methodology

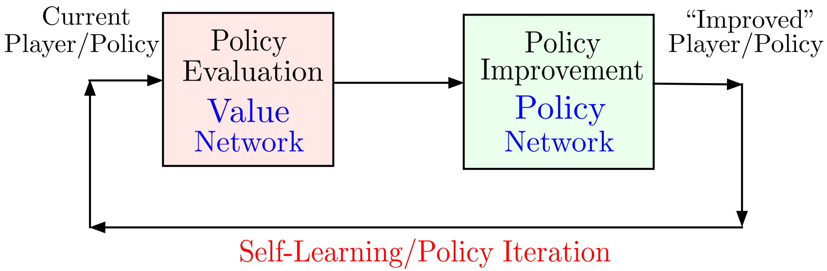
- We are aiming to understand this methodology, so it applies far more generally
- We focus on connections with control system design (MPC and adaptive control), and on discrete optimization

On-Line Play in AlphaZero/AlphaGo/TD-Gammon: Approximation in Value Space (Also Called “On-Line Tree Search")



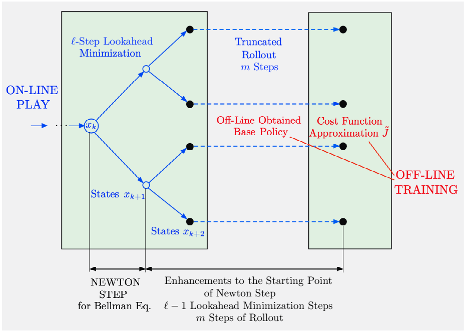
- On-line play uses the results of off-line training, which are: A position evaluator and a base player
- It aims to improve the base player by:
  - Searching forward for several moves through the lookahead tree
  - Simulating the base player for some more moves at the tree leaves
  - Approximating the effect of future moves by using the terminal position evaluation
  - Calculating the “values" of the available moves at the root and playing the best move
- Similarities with Model Predictive Control (MPC) architecture

Off-Line Training in AlphaZero: Approximate Policy Iteration (PI) Using Self-Generated Data



- The current player is used to train an improved player, and the process is repeated
- The current player is "evaluated" by playing many games
- Its evaluation function is represented by a value neural net through training
- The current player is "improved" by using a form of approximate multistep lookahead minimization, called Monte-Carlo Tree Search (MCTS)
- The "improved player" is represented by a policy neural net through training
- TD-Gammon uses similar PI algorithm for off-line training of a value network (does not use MCTS and does not use a policy network)

Some Major Empirical Observations



ON-LINE PLAY
ℓ-Step Lookahead Minimization
x_k
States x_{k+1}
States x_{k+2}
Truncated Rollout m Steps
Off-Line Obtained Base Policy
Cost Function Approximation $\tilde{J}$
OFF-LINE TRAINING
NEWTON STEP for Bellman Eq.
Enhancements to the Starting Point of Newton Step
$\ell - 1$ Lookahead Minimization Steps
m Steps of Rollout

The AlphaZero on-line player plays much better than the off-line-trained player

TD-Gammon plays much better with truncated rollout than without rollout (Tesauro, 1996)

We will aim for explanations, insights, and generalizations through abstract Bellman operators, visualization, and the

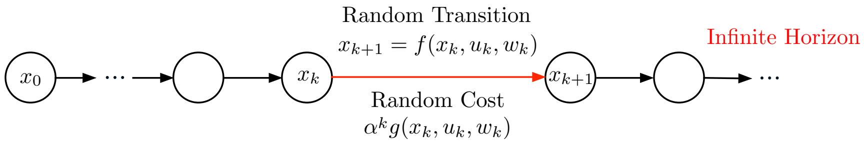
central role of Newton's method

Principal Viewpoints of this Talk

- On-line play is a single step of Newton's method for solving the Bellman equation (or Newton-SOR in case of multistep lookahead and/or truncated rollout)
- Off-line training provides the start point for the Newton step
- On-line play is the real workhorse ... off-line training plays a secondary role. A major reason: On-line play is an exact Newton step. It is not degraded by NN approximations
- Imperfections/differences in off-line training affect the start point, but are washed out by the (superlinear) Newton step
- A cultural difference that we will aim to bridge:
  - Reinforcement Learning/AI research is focused largely on off-line training issues (except in the special case of armed bandit problems)
  - Model Predictive Control research is focused largely on on-line play and stability issues
- Application to adaptive control (changing system parameters): It's still an exact Newton step applied to an on-line estimated Bellman equation
- All of this applies in great generality through the power of abstract DP (arbitrary state and control spaces, stochastic, deterministic, hybrid systems, multiagent systems, minimax, finite and infinite horizon, discrete optimization)

Outline

Infinite Horizon Problems



Infinite number of stages, and stationary system and cost

- System $x_{k+1} = f(x_k, u_k, w_k)$ with state, control, and random disturbance
- Stationary policies $x \mapsto \mu(x)$ satisfying a control constraint $\mu(x) \in U(x)$ for all x
- Cost of stage k: $\alpha^k g\big(x_k, \mu(x_k), w_k\big)$; $0 < \alpha \leq 1$ is the discount factor
- Cost of a policy μ: The limit as $N \to \infty$ of the N-stage costs

$$J_\mu(x_0) = \lim_{N \to \infty} E_{w_k} \left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu(x_k), w_k\big) \right\}$$

- Optimal cost function $J^*(x_0) = \min_\mu J_\mu(x_0)$
- Discounted problems: $\alpha < 1$ and g is bounded (the “nice" case)
- Stochastic shortest path problems: $\alpha = 1$ and special cost-free termination state t
- Control/MPC-type problems: Deterministic, $g \geq 0$, termination state is $t = 0$

Infinite Horizon Problems - Main (Exact DP) Theory

J^* satisfies Bellman's equation:

$$J^*(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J^*\big(f(x,u,w)\big)\Big\}, \qquad \text{for all states } x \quad \text{(uniquely ??)}$$

Optimality condition: If $\mu^*(x)$ attain the min in the Bellman equation for every x, the policy μ^* is optimal (??)

Value iteration (VI): Generates cost function sequence $\{J_k\}$

$$J_k(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J_{k-1}\big(f(x,u,w)\big)\Big\}, \qquad J_0 \text{ is “arbitrary" (??)}$$

Policy Iteration (PI): Generates sequences of policies $\{\mu^k\}$ and their cost functions $\{J_{\mu^k}\}$; μ^0 is “arbitrary" (??)

The typical iteration starts with a policy μ and generates a new policy $\tilde{\mu}$ in two steps:

- Policy evaluation step: Computes the cost function J_μ (possibly w/ approximations)
- Policy improvement step: Computes the improved policy $\tilde{\mu}$ using

$$\tilde{\mu}(x) \in \arg\min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J_\mu\big(f(x,u,w)\big)\Big\}$$

On-Line Approximation in Value Space

- Replace J^* with an approximation $\tilde J$ in Bellman's equation

$$\boxed{\text{At } x} \longrightarrow \min_{u\in U(x)} E\Big\{\overbrace{g(x,u,w)}^{\text{First Step}} + \overbrace{\alpha\tilde J\big(f(x,u,w)\big)}^{\text{“Future”}}\Big\}$$

One-Step Lookahead

$$\boxed{\text{At } x_k} \longrightarrow \min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{\overbrace{g(x_k,u_k,w_k) + \sum_{i=k+1}^{k+\ell-1} \alpha^{i-k} g\big(x_i,\mu_i(x_i),w_i\big)}^{\text{First } \ell \text{ Steps}} + \overbrace{\alpha^{\ell}\tilde J(x_{k+\ell})}^{\text{“Future”}}\right\}$$

Multistep Lookahead

- Defines a lookahead/“greedy" policy $\tilde\mu$ with $\tilde\mu(x_k)$ the minimizing u_k above

KEY NEW FACT: $J_{\tilde\mu}$ is the result of a Newton step to solve Bellman Eq. starting from $\tilde J$ (Newton-SOR step for multistep lookahead $\ell > 1$). The error decreases SUPERLINEARLY

$$\frac{J_{\tilde\mu} - J^*}{\tilde J - J^*} \to 0 \qquad \text{as } \tilde J \to J^*$$

An Abstract DP Viewpoint: Bellman Operators and Bellman Equations (Abstract DP Book, 3rd ed., 2022)

$$(T_\mu J)(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big)\Big\}$$

$$(TJ)(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J\big(f(x,u,w)\big)\Big\} = \min_{\mu} (T_\mu J)(x)$$

They define the Bellman equations $J_\mu = T_\mu J_\mu, \quad J^* = TJ^*$

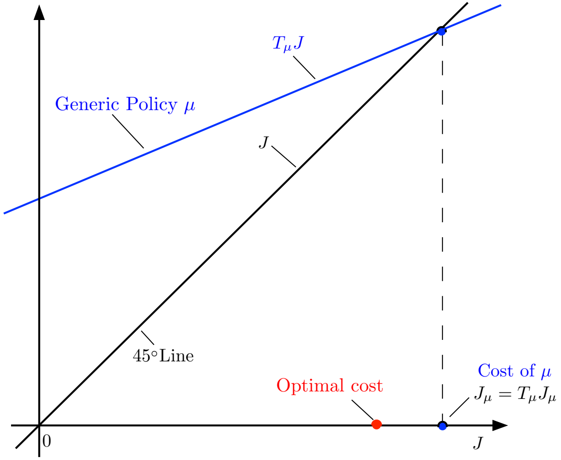
T_μ and T transform real-valued functions J into functions $T_\mu J$ and TJ (assumed real-valued for this talk)

- For each fixed x, $(T_\mu J)(x)$ and $(TJ)(x)$ are functions of J
- T_μ is monotone and linear
- T is monotone and “concave", i.e., $(TJ)(x)$ is a concave function of J for each fixed x
- Example: For a 2-state system, $(TJ)(1)$ and $(TJ)(2)$ are real-valued functions of the vector $J = \big(J(1), J(2)\big) \in \Re^2$
- For infinite-dimensional state space, T_μ and T are infinite-dimensional operators (map infinite dimensional function space to itself)
- Our approach: Use visualization along 1-D slices of the graphs of the operators; verify/generalize the results of the visualization with math analysis

μ-Bellman Operator in One Dimension Through J_μ

$$(T_\mu J)(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big)\Big\} \quad \text{(linear monotone)}$$

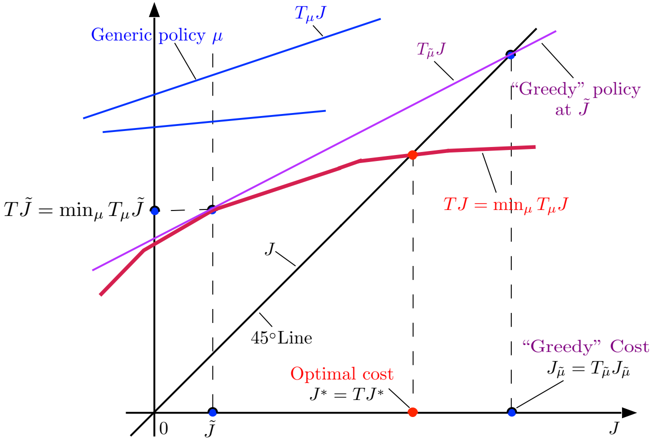
μ-Bellman equation: $J_\mu = T_\mu J_\mu$



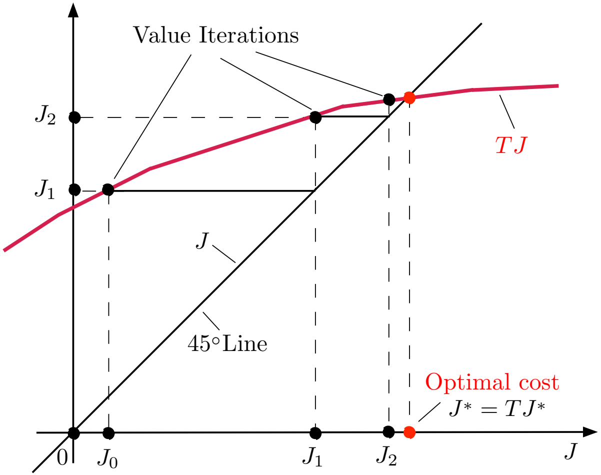
Min-Bellman Operator in One Dimension Through J^*

$$(TJ)(x) = \min_{u \in U(x)} E_w\Big\{ g(x,u,w) + \alpha J\big(f(x,u,w)\big)\Big\} = \min_{\mu} (T_\mu J)(x) \quad \text{(concave monotone)}$$

Min-Bellman equation: $J^* = TJ^*$

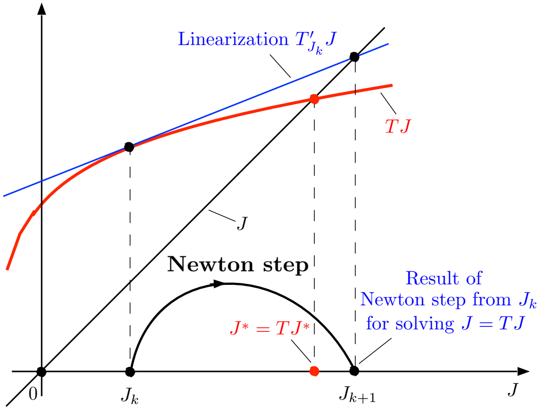

Generic policy μ
$T_\mu J$
$T_{\tilde\mu} J$
"Greedy" policy at $\tilde J$
$T\tilde J = \min_\mu T_\mu \tilde J$
$TJ = \min_\mu T_\mu J$
J
45° Line
Optimal cost $J^* = TJ^*$
"Greedy" Cost $J_{\tilde\mu} = T_{\tilde\mu} J_{\tilde\mu}$
0
$\tilde J$
J

Visualization of Value Iteration: $J_{k+1} = TJ_k$



Convergence $J_k \to J^*$ depends on J_0 and the “slope" of T (e.g., whether T is a contraction)

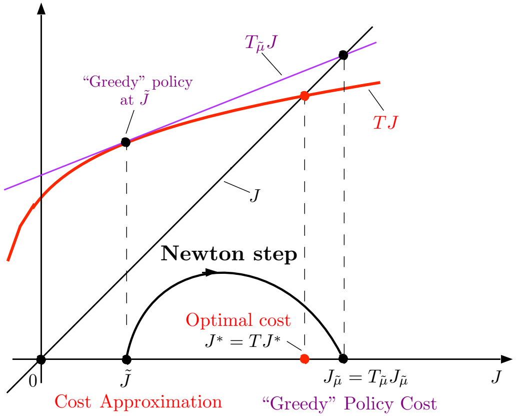
Newton's Method for Solving Generic Fixed Point Equation $J = TJ$


Linearization $T'_{J_k} J$
TJ
J
Newton step
Result of Newton step from J_k for solving $J = TJ$
$J^* = TJ^*$
0
J_k
J_{k+1}
J

It is an iterative method that generates a sequence $\{J_k\}$. The typical iteration:

- Given J_k, "linearize" T at J_k: Replace TJ by the linearization $T'_{J_k} J$
- Solve the linearized fixed point problem $J = T'_{J_k} J$
- The solution of the linearized fixed point problem is the next iterate J_{k+1}
- Do we need differentiability of T? Answer: NO (concavity of T is enough)

Back to DP: One-Step Lookahead - Newton Step Interpretation

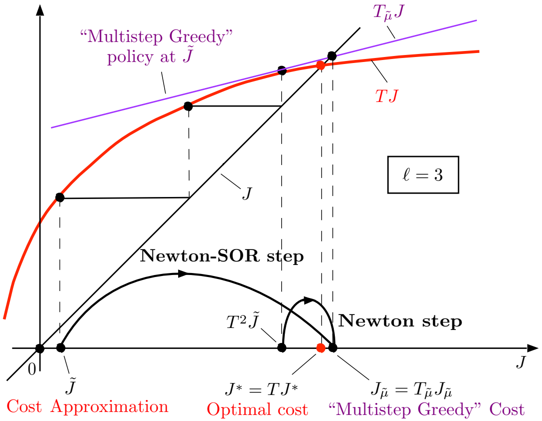


$T_{\tilde\mu} J$
"Greedy" policy at $\tilde J$
TJ
J
Newton step
Optimal cost $J^* = TJ^*$
0
$\tilde J$
$J_{\tilde\mu} = T_{\tilde\mu} J_{\tilde\mu}$
J
Cost Approximation
"Greedy" Policy Cost

$$(T\tilde J)(x) = \min_{\mu} (T_\mu \tilde J)(x) = (T_{\tilde\mu}\tilde J)(x) \quad \text{(linearization of } T \text{ at } \tilde J \text{ yields "greedy" } \tilde\mu)$$

- This is a key new insight with important ramifications:
- The Newton step smooths out starting point variations (lots of empirical evidence)
- Local error decreases superlinearly (much better than current theory suggests)

Multistep Lookahead - Newton-SOR Step Interpretation

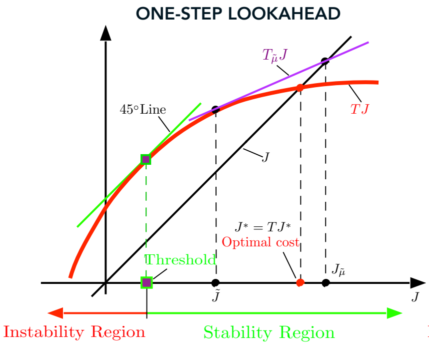


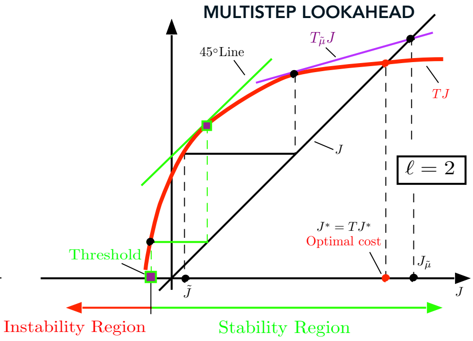
$(T^\ell \tilde J)(x) = \min_\mu (T_\mu T^{\ell-1}\tilde J)(x) = (T_{\tilde\mu} T^{\ell-1}\tilde J)(x)$ (linearization of T at $T^{\ell-1}\tilde J$ yields $\tilde\mu$)

Solution of the linearized equation $J = T_{\tilde\mu}J$ yields the cost function $J_{\tilde\mu}$ of $\tilde\mu$

Newton-SOR converges faster than pure Newton, but is more time-consuming

Stability: How do we Know that the “Greedy" Policy is Stable?





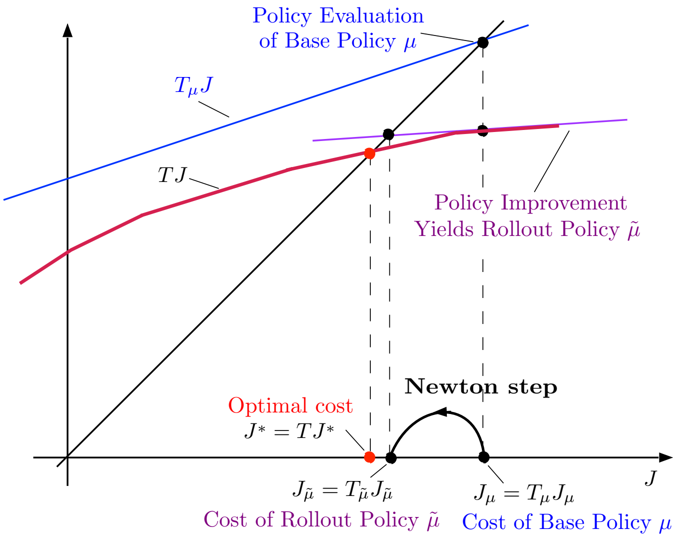
A policy μ is called stable if $J_\mu(x) < \infty$ for all x (a very general definition)

True if T_μ has “slope" < 1 (i.e., T_μ is a contraction)

Region of stability: The set of $\tilde{J}$ for which the “greedy" policy $\tilde{\mu}$ is stable

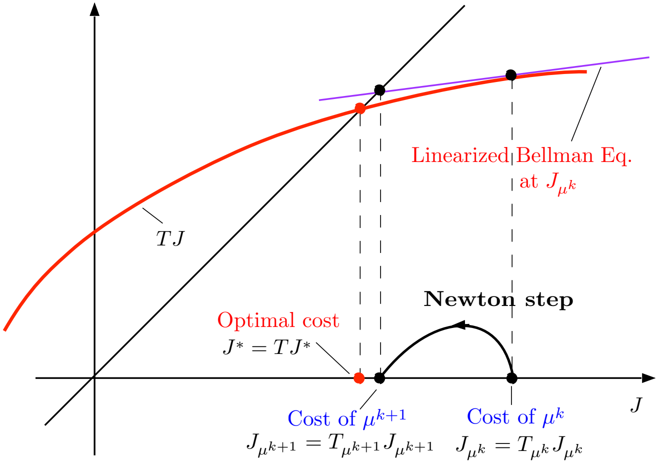
- Depends on the length of lookahead - longer lookahead promotes stability
- It makes sense to try to push $\tilde{J}$ towards some J_μ with μ stable (rollout idea)

Rollout: A Newton Step Starting from $\tilde{J} = J_\mu$, where μ is a Stable Policy


Policy Evaluation of Base Policy μ
$T_\mu J$
TJ
Policy Improvement Yields Rollout Policy $\tilde{\mu}$
Newton step
Optimal cost $J^* = TJ^*$
$J_{\tilde{\mu}} = T_{\tilde{\mu}} J_{\tilde{\mu}}$
Cost of Rollout Policy $\tilde{\mu}$
$J_\mu = T_\mu J_\mu$
Cost of Base Policy μ
J

Rollout with a stable policy μ yields a (better) stable policy $\tilde{\mu}$

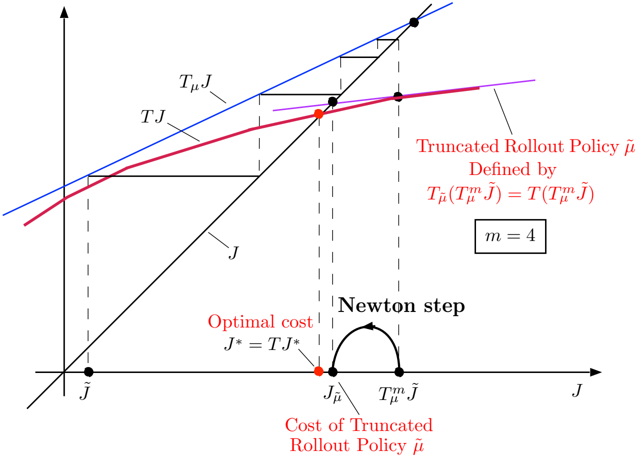
Policy Iteration is Repeated Rollout - Starting from a Stable Policy it Produces a Sequence of Stable Policies $\{\mu^k\}$



TJ
Linearized Bellman Eq. at J_{μ^k}
Newton step
Optimal cost $J^* = TJ^*$
Cost of μ^{k+1} $J_{\mu^{k+1}} = T_{\mu^{k+1}} J_{\mu^{k+1}}$
Cost of μ^k $J_{\mu^k} = T_{\mu^k} J_{\mu^k}$
J

- Pure form of PI is Newton's method (known for special cases, Kleinman 1968 ++)

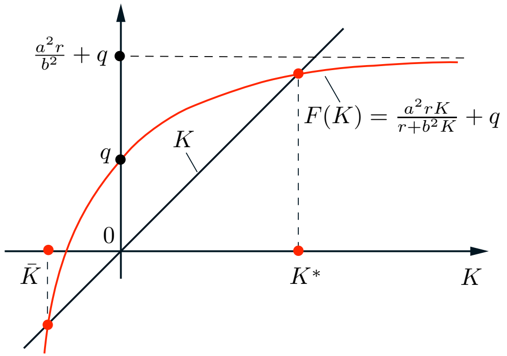
Rollout with Approximate Policy Evaluation (Truncated Rollout - A Single “Optimistic" PI)



Truncated rollout with $\ell > 1$-step lookahead min is similar; $\ell + m$ is what matters

Truncated rollout is an economical substitute for multistep lookahead (e.g., TD-Gammon)

Linear Quadratic Problems: Riccati Instead of Bellman Operators



Riccati operator F is the restriction of the Bellman operator to the subspace of quadratics

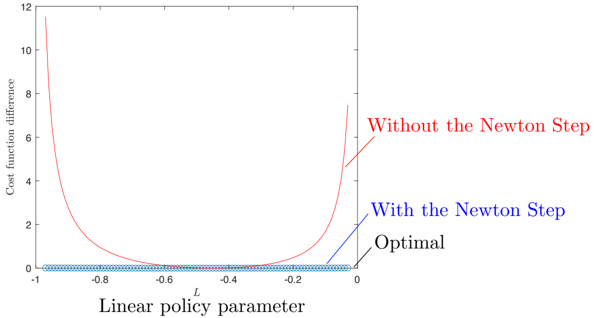
Linear system $x_{k+1} = ax_k + bu_k$. Cost $g(x, u) = qx^2 + ru^2$, $q, r > 0$, $\alpha = 1$

- $J^*(x) = K^* x^2$; K^* solves the Riccati Eq. $K = F(K)$
- Riccati Eq. has K^* as its unique positive solution
- Visualizations extend - LQ problems are useful for experimentation and insights

A Common Question: Why Not Use Approximation in Policy Space? (Just Train a Policy Network to Emulate the On-Line Player)

Pure approx. in policy space (policy gradient, random search, etc) is flawed

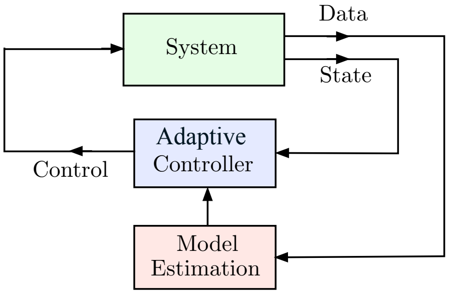
It lacks the exact Newton step, which corrects (superlinearly) the errors of off-line training



A one-dimensional linear quadratic example (with known and fixed model)

Compare a parametrized suboptimal linear policy $\mu(x) = Lx$ with and without one-step lookahead/Newton step

Adaptive Control - Changing System, On-line System Identification



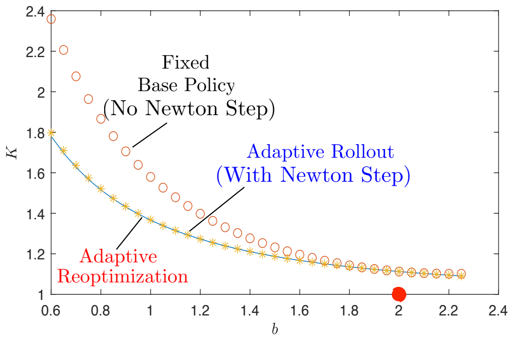
Classical indirect adaptive control (1960s+, extensive book literature)

Simply reoptimizes the controller, when the estimated model changes ... but this may be a difficult/time-consuming reoptimization

Faster alternative: Indirect adaptive control by rollout with a (robust) policy

- Use rollout in place of reoptimization - this is simpler (use the current model estimate for lookahead minimization and a nominal/robust base policy for rollout)
- Capitalizes on the fast convergence of the Newton step

Adaptive Control by Rollout: A Linear Quadratic Example



$$x_{k+1} = x_k + bu_k, \qquad g(x,u) = x^2 + 0.5u^2$$

- We use one-step lookahead and rollout with the base policy that is optimal for the nominal value $b = 2$
- We change the system parameter b in the range $[0.6, 2.4]$
- Using a “robust" controller without the Newton step is often flawed
- Using a “robust" controller as base policy with the Newton step corrects the flaw

An Alternative Adaptive Control Formulation: A POMDP Approach from the 70s-80s

Deterministic system $x_{k+1} = f(x_k, \theta, u_k)$, $\theta \in \{\theta^1, \ldots, \theta^m\}$: unknown parameter

- View θ as part of an augmented state (x_k, θ) that is partially observed
- Belief state: $b_{k,i} = P\{\theta = \theta^i \mid x_0, \ldots, x_k\}$, $i = 1, \ldots, m$, (estimated on-line)
- Bellman equation for optimal cost function J_k^*:

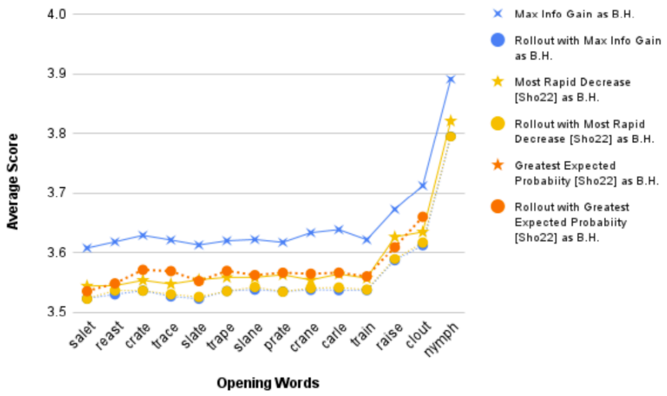
$$J_k^*(x_0, \ldots, x_k) = \min_{u_k} \sum_{i=1}^{m} b_{k,i}\left(g(x_k, \theta^i, u_k) + J_{k+1}^*\left(x_0, \ldots, x_k, f(x_k, \theta^i, u_k)\right)\right)$$

- Approximation in value space: Use approximation $\tilde{J}^i\left(f(x_k, \theta^i, u_k)\right)$ in place of $J_{k+1}^*\left(x_0, \ldots, x_k, f(x_k, \theta^i, u_k)\right)$. Minimize over u_k to obtain "greedy" policy
- Example 1: $\tilde{J}^i$ is the cost function of the optimal policy corresponding to θ^i
- Example 2: $\tilde{J}^i$ is the cost function of a known policy assuming $\theta = \theta^i$ (this is rollout)

An example: The popular Wordle puzzle

- Try to find a mystery word θ using successive 5-letter guess words
- View θ as part of an augmented state (x_k, θ) that is partially observed
- Joint work in preparation with S. Bhambri and A. Bhattacharjee (ASU)

Rollout Results for Wordle with Three Different Base Policies



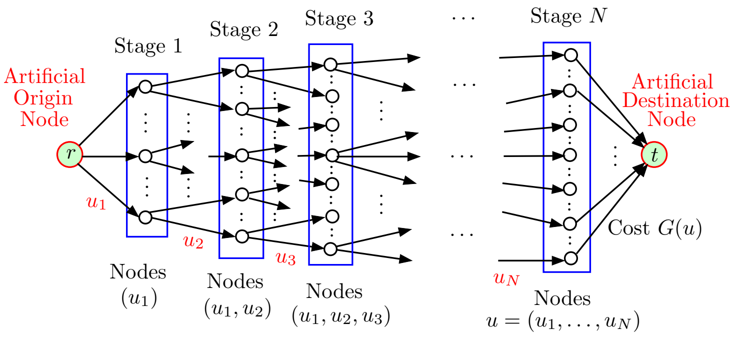
Rollout Performance = 3.5231 vs the Optimal = 3.5084 average # of guesses

Within $<$ 0.5% more guesses from the optimal policy - On-line answer within 1-3 secs

IT SCALES WITH PROBEM SIZE

The Newton step washes out the performance differences of the base policies

Generality - Approximation in Value Space Applies to Continuous and Discrete Spaces/Integer Programming



Minimizing $G(u)$ over $u = (u_1, \ldots, u_N) \in U$, where u_i are discrete components

- It admits a (continuous space) Bellman equation and a Newton step interpretation
- Approximation in value space applies; e.g., $\tilde{J}(u_1, \ldots, u_k)$ is cost of a heuristic applied after $u_1, \ldots, u_k$ have been chosen
- Rollout approach is simple and very successful in practice
- Off-line training with data is possible

Concluding Remarks

- There is much to be gained by using on-line play on top of off-line training
- Using just off-line training without on-line play may not work well
  - On-line play uses an exact Newton step (not subject to training errors), and can deal with changing system parameters
- Using just on-line play without off-line training misses out on performance
  - Off-line training can produce good starting points for the Newton step
- The role of Newton's method is central - this is a new insight that can guide both analysis and algorithmic design
- The Newton step is exact ... all the approximation goes into the starting point for the Newton step (which washes out training method differences and errors)
- The cultural divide between RL/AI and control can be bridged by combining off-line training and on-line play
- MPC uses a very similar architecture to AlphaZero; can benefit from RL/AI ideas
- We can approach adaptive control through rollout
- Generality: Arbitrary state and control spaces, discrete optimization applications, multiagent versions (see the 2020 rollout/distributed RL book)
- There are exceptional behaviors waiting for clarification by analysis

Some Words of Optimism

- The successes of RL and of MPC are solid reasons for optimism
- More success can be expected by combining ideas from both RL/AI and MPC/adaptive control cultures
- On-line long lookahead/rollout can be a computational bottleneck ...
- But massive computational power and distributed computation can mitigate the bottleneck, and allow more sophisticated on-line play strategies
- There is an exciting journey ahead!

Thank you!